Aspects of Enumeration and Generation with a String Automata Representation [*][†]

Marco Almeida
mfa@ncc.up.pt

Nelma Moreira
nam@ncc.up.pt

Rogério Reis
rvr@ncc.up.pt

DCC-FC & LIACC, Universidade do Porto
R. do Campo Alegre 823, 4150 Porto, Portugal

Abstract

In general, the representation of combinatorial objects is decisive for the feasibility of several enumerative tasks. In this work, we show how a (unique) string representation for (complete) initially-connected deterministic automata (ICDFA's) with n states over an alphabet of k symbols can be used for counting, exact enumeration, sampling and optimal coding, not only the set of ICDFA's but, to some extent, the set of regular languages. An exact generation algorithm can be used to partition the set of ICDFA's in order to parallelize the counting of minimal automata (and thus of regular languages). We present also a uniform random generator for ICDFA's that uses a table of pre-calculated values. Based on the same table it is also possible to obtain an optimal coding for ICDFA's.

Keyword: regular languages, initially-connected deterministic finite automata, enumeration, random generation

1 Introduction

In general, the representation of combinatorial objects is decisive for the feasibility of several enumerative tasks. In this work, we show how a (unique) string representation for (complete) initially-connected deterministic automata (ICDFA's) with n states over an alphabet of k symbols can be used for counting, exact enumeration, sampling and optimal coding, not only the set of ICDFA's but, to some extent, the set of regular languages. The key fact is that string representations are characterized by a set of rules that allow an exact and ordered generation of all its elements. An exact generation algorithm can be used to partition the set of ICDFA's in order to parallelize the counting of minimal automata, and thus of regular languages. With the same set of rules it is possible to design a uniform random generator for ICDFA's that uses a table of pre-calculated values

[*] Work partially funded by Fundação para a Ciência e Tecnologia (FCT) and Program POSI.

[†] This paper was presented at the 8th Workshop on Descriptional Complexity of Formal Systems, DCFS'06.

(as usual in combinatorial decomposition approaches). Based on the same table it is also possible to obtain an optimal coding for ICDFA's (with or without final states).

In the next section, some definitions and notation are introduced. In Section 3 we review the string representation of non-isomorphic ICDFA$_\emptyset$'s (i.e., ICDFA's without final states), and how it can be used to generate and enumerate all ICDFA's. We also relate those methods to the ones presented by Champarnaud and Paranthöen in [CP05], by giving a new enumerative result. In Section 4, we briefly describe the implementation of a generator algorithm for ICDFA$_\emptyset$'s. Section 5 presents the methods for parallelizing the counting of languages by slicing the universe of ICDFA$_\emptyset$'s and some experimental results are given. A uniform random generator for ICDFA$_\emptyset$'s is described in Section 6 along with some experimental results and statistical tests. Using the recurrence formulae defined in Section 6, we show in Section 7 how we can associate an integer with an ICDFA$_\emptyset$'s and vice-versa. Section 8 concludes with final remarks.

2 Preliminaries

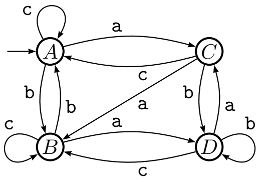
Given two integers $m < n$ we represent the set $\{i \in \mathbb{N} \mid m \leq i \leq n\}$ by $[m,n]$. A *deterministic finite automaton* (DFA) $\mathcal{A}$ is a quintuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ the alphabet, i.e, a non-empty finite set of symbols, $\delta : Q \times \Sigma \to Q$ is the transition function, q_0 the initial state and $F \subseteq Q$ the set of final states. The *size of the automaton* is given by $|Q|$. We assume that the transition function is total, so we consider only *complete* DFA's. As we are not interested in the labels of the states, we can represent them by an integer $i \in [0, |Q| - 1]$. The transition function δ extends naturally to $\Sigma^\star$. A DFA is *initially-connected*[1] (ICDFA) if for each state $q \in Q$ there exists a string $x \in \Sigma^\star$ such that $\delta(q_0, x) = q$. The *structure* of an automaton (Q, Σ, δ, q_0) denotes a DFA without its final state information and is referred to as a DFA$_\emptyset$. For each structure, there will be 2^n DFA's, if $|Q| = n$. We denote by ICDFA$_\emptyset$ the structure of an ICDFA. Two DFA's $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and $\mathcal{A}' = (Q', \Sigma, \delta', q_0', F')$ are called *isomorphic* (by states) if there exists a bijection $f : Q \to Q'$ such that $f(q_0) = q_0'$ and for all $\sigma \in \Sigma$ and $q \in Q$, $f(\delta(q, \sigma)) = \delta'(f(q), \sigma)$. Furthermore, for all $q \in Q$, $q \in F$ if and only if $f(q) \in F'$. The *language* accepted by a DFA $\mathcal{A}$ is $L(\mathcal{A}) = \{x \in \Sigma^\star \mid \delta(q_0, x) \in F\}$. Two DFA are *equivalent* if they accept the same language. Obviously, two isomorphic automata are equivalent, but two non-isomorphic automata may also be equivalent. A DFA $\mathcal{A}$ is *minimal* if there is no DFA $\mathcal{A}'$ with fewer states equivalent to $\mathcal{A}$. Trivially a minimal DFA is an ICDFA. Minimal DFA's are unique up to isomorphism. Domaratzki et al. [DKS02] gave some asymptotic estimates and explicit computations of the number of distinct languages accepted by finite automata with n states over an alphabet of k symbols. Given n and k, they denoted by $f_k(n)$ the number of pairwise non-isomorphic minimal DFA's and by $g_k(n)$ the number of distinct languages accepted by DFA's, where $g_k(n) = \sum_{i=1}^{n} f_k(i)$.

[1] Also called *accessible*.

3 Strings for ICDFA's

Reis et al. [RMA05] presented a unique string representation for non-isomorphic ICDFA$_\emptyset$'s. In this section, we briefly review this representation and how it can be used to generate and enumerate all ICDFAs. We also give a new enumerative result and relate this representation to the one presented by Champarnaud and Paranthöen in [CP05].

Given a complete DFA$_\emptyset$ (Q, Σ, δ, q_0) with $|Q| = n$ and $|\Sigma| = k$, consider a total order $<$ over Σ. We can define a canonical order over the set of the states by exploring the automaton in a breadth-first way choosing at each node the outgoing edges in the order considered for Σ. If we restrict this representation to ICDFA$_\emptyset$'s, then this representation is unique and defines an order over the set of its states. For instance, consider the following ICDFA$_\emptyset$ and consider the alphabetic order in $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.



The states ordering is A,C,B,D and $[1, 2, 0, 2, 3, 0, 3, 0, 2, 1, 3, 2]$ is its string representation. Formally, let $\Sigma = \{\sigma_i \mid i \in [0, k-1]\}$, with $\sigma_0 < \sigma_1 < \cdots < \sigma_{k-1}$. Given an ICDFA$_\emptyset$ (Q, Σ, δ, q_0) with $|Q| = n$, the representing string is of the form $(s_i)_{i \in [0,kn-1]}$ with $s_i \in [0, n-1]$ and $s_i = \delta(\lfloor i/k \rfloor, \sigma_{i \bmod k})$.

Let $(s_i)_{i \in [0,kn-1]}$ with $s_i \in [0, n-1]$ be a string satisfying the following conditions:

$$(\forall m \in [2, n-1])(\forall i \in [0, kn-1])(s_i = m \Rightarrow (\exists j \in [0, i-1])\, s_j = m-1). \qquad (\mathbf{R1})$$

$$(\forall m \in [1, n-1])(\exists j \in [0, km-1])\, s_j = m. \qquad (\mathbf{R2})$$

In [RMA05] the following theorem was proved.

Theorem 1 *There is a one-to-one mapping between $(s_i)_{i \in [0,kn-1]}$ with $s_i \in [0, n-1]$ satisfying rules **R1** and **R2**, and the non-isomorphic* ICDFA$_\emptyset$*'s with n states, over an alphabet Σ of size k.*

We note that this string representation can be extended to non-complete ICDFA$_\emptyset$'s, by representing all missing transitions with the value -1. In this case, rules **R1** and **R2** remain valid, and we can assume that the transitions from this state are into itself. However for enumeration and generation purposes we do not consider non-complete ICDFA$_\emptyset$'s.

In order to have an algorithm for the enumeration and generation of ICDFA$_\emptyset$'s, instead of rules **R1** and **R2** an alternative set of rules were used. For $n = 1$ there is only one (non-isomorphic) ICDFA$_\emptyset$ for each $k \geq 1$, so we assume in the following that $n > 1$. In a string representing an ICDFA$_\emptyset$, let $(f_j)_{j \in [1,n-1]}$ be the sequence of indexes of the first occurrence of each state label j. For explanation purposes, we call those indexes *flags*. It is easy to see that (**R1**) and (**R2**) correspond respectively to (**G1**) and (**G2**):

$$(\forall j \in [2, n-1])(f_j > f_{j-1}); \qquad (\mathbf{G1})$$

$$(\forall m \in [1, n-1])\ (f_m < km). \qquad (\mathbf{G2})$$

This means that $f_1 \in [0, k-1]$, and $f_{j-1} < f_j < kj$ for $j \in [2, n-1]$. We begin by counting the number of sequences of flags allowed.

Theorem 2 *Given k and n, the number of sequences $(f_j)_{j\in[1,n-1]}$, $F_{k,n}$, is given by*

$$F_{k,n} = \sum_{f_1=0}^{k-1} \sum_{f_2=f_1+1}^{2k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1} 1 = \binom{kn}{n} \frac{1}{(k-1)n+1} = C_n^{(k)};$$

where $C_n^{(k)}$ are the (generalised) Fuss-Catalan numbers.

Proof 1 *The first equality follows directly from the definition of the $(f_j)_{j\in[1,n-1]}$. For the second, note that $C_n^{(k)}$ enumerates k-ary trees with n internal nodes, $\mathcal{T}_n^k$ (see for instance [SF96]). In particular, for $k = 2$, C_n^2 are exactly the Catalan numbers that count binary trees with n internal nodes. This sequence appears in Sloane [Slo03] as **A00108** and for $k = 3$ and $k = 4$ as **A001764** and **A002293** sequences, respectively. So it suffices to give a bijection between these trees and the sequences of flags. Recall that a k-ary tree is an external node or an internal node attached to an ordered sequence of k, k-ary sub-trees.*

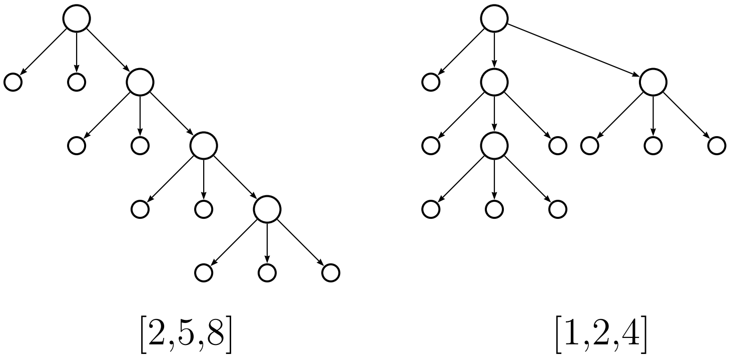


Figure 1: Two 3-ary trees with 4 internal nodes and the correspondent sequence of flags.

*Let $\mathcal{T}_n^k$ be a k-ary tree and let $<$ be a total order over Σ. For each internal node i of $\mathcal{T}_n^k$ its outgoing edges can be ordered left-to-right and attached a unique symbol of Σ according to $<$. Considering a breadth-first, left-to-right, traversal of the tree and ignoring the root node (that is considered the 0-th internal node), we can represent $\mathcal{T}_n^k$, uniquely, by a bitmap where a 0 represents an external node and a 1 represents an internal node. As the number of external nodes are $(k-1)n+1$, the length of the bitmap is kn. Moreover the $j+1$-th block of k bits corresponds to the children of the j-th internal node visited, for $j \in [0, n-1]$. For example, the bitmaps of the trees in Figure 1 are $[0,0,1,0,0,1,0,0,1,0,0,0]$ and $[0,1,1,0,1,0,0,0,0,0,0,0]$, respectively. The positions of the 1's in the bitmaps correspond to a sequence of flags, $(f_i)_{i\in[1,n-1]}$, i.e., f_i corresponds to the number of nodes visited before the i-th internal node (excluding the root node). It is obvious that $(f_i)_{i\in[1,n-1]}$ verify **G1**. For **G2**, note that for the each internal node the outdegree of the previous internal nodes is k. Conversely, given a sequence of flags $(f_j)_{j\in[1,n-1]}$, we construct the bitmap such that b_{f_i}=1 for $i \in [1, n-1]$ and $b_j = 0$ for the remaining values, for $j \in [0, kn-1]$. As above, for the representation of the $j+1$-th internal node, $\lfloor f_j/k \rfloor$ gives the parent and $f_j \bmod k$ gives its position between its siblings (in breadth-first, left-to-right traversal).*

To generate all the ICDFA$_\emptyset$'s, for each allowed sequence of flags $(f_j)_{j\in[1,n-1]}$, all the remaining symbols s_i can be generated according to the following rules:

$$i < f_1 \Rightarrow s_i = 0; \tag{\textbf{G3}}$$

$$(\forall j \in [1, n-2])(f_j < i < f_{j+1} \Rightarrow s_i \in [0, j]); \tag{\textbf{G4}}$$

$$i > f_{n-1} \Rightarrow s_i \in [0, n-1]. \tag{\textbf{G5}}$$

In [RMA05] a simple combinatorial argument was given to show that

Theorem 3 *The number of strings $(s_i)_{i\in[0,kn-1]}$ representing* ICDFA$_\emptyset$*'s with n states over an alphabet of k symbols is given by*

$$B_{k,n} = \sum_{f_1=0}^{k-1} \sum_{f_2=f_1+1}^{2k-1} \sum_{f_3=f_2+1}^{3k-1} \cdots \sum_{f_{n-1}=f_{n-2}+1}^{k(n-1)-1} \prod_{i=2}^{n} i^{f_i - f_{i-1} - 1}; \tag{1}$$

where $f_n = kn$.

In Section 6 we give other recursive definition that is more adequate for tabulation.

3.1 Analysis of the Champarnaud et al. Method

Champarnaud and Paranthoën in [CP05, Par04], generalizing work of Nicaud [Nic00] presented a method to generate and enumerate ICDFA$_\emptyset$'s, although not giving an explicit and compact representation for them, as the string representation used here. An order $<$ over $\Sigma^\star$ is a *prefix order* if $(\forall x \in \Sigma^\star)(\forall \sigma \in \Sigma) x < x\sigma$. Let $\mathcal{A}$ be an ICDFA$_\emptyset$ over Σ with k symbols and n states. Given a prefix order in $\Sigma^\star$, each automaton state is ordered according to the first word $x \in \Sigma^\star$ that reaches it in a simple path from the initial state. The sets of this words $\{\mathcal{P}\}$ are in bijection with k-ary trees with n internal nodes, and therefore to the set of sequences of flags, in our representation[2]. Then it is possible to obtain a valid ICDFA$_\emptyset$ by adding other transitions in a way that preserves the previous state labelling. For the generation of the sets $\mathcal{P}$ it is used another set of objects that are in bijection with k-ary trees with n internal nodes and are called generalised tuples. The number of ICDFA$_\emptyset$'s is computed using recursive formulae associated with generalized tuples, akin the ones we present in Section 6.

4 Generating ICDFA$_\emptyset$'s

In this section, we present a method to generate all ICDFA$_\emptyset$'s, given k and n. We start with an initial string, and then consecutively iterate over all allowed strings until the last one is reached. The main procedure is the one that given a string returns the *next* legal one. For each k and n, the first ICDFA$_\emptyset$ is represented by the string $0^{k-1}10^{k-1}\ldots(n-1)0^k$ and the last is represented by $12\ldots(n-1)(n-1)^{(k-1)n+1}$. According to the rules **G1- G5**, we first generate a sequence of flags, and then, for each one, the set of strings representing the ICDFA$_\emptyset$'s in lexicographic order. The algorithm to generate the next sequence of flags is the following, where the initial sequence of flags is $(ki-1)_{i\in[1,n-1]}$:

[2] Indeed our order on the states induces a prefix order in $\Sigma^\star$.

```
def nextflags($i$):
  if $i$==1 then $f_i = f_i - 1$
  else
      if ($f_i$−1 == $f_{i-1}$) then
          $f_i = k * i - 1$
          nextflags($i - 1$)
      else $f_i = f_i - 1$
```

To generate a new sequence, we must call **nextflags(n-1)**. Given the rules **G1** and **G2** the correctness of the algorithm is easily proved. When a new sequence of flags is generated, the first ICDFA$_\emptyset$ is represented by a string with 0s in all other positions (i.e., the lower bounds in rules **G3**–**G5**). The following strings, with the same sequence of flags, are computed lexicographically using the procedure **nexticdfa**, called with $a = n - 1$ and $b = k - 1$:

```
def nexticdfa($a, b$):
    $i = a * k + b$
        if $a < n - 1$ then
            while $i \in (f_j)_{j\in[1,n-1]}$:
                for $k = i + 1$ to $kn - 1$:
                    if $k \notin (f_j)_{j\in[1,n-1]}$ then $s_k = 0$
                $b = b - 1$
                $i = i - 1$
        $f_j$ = the nearest flag not exceeding $i$
        if $s_i == s_{f_j}$ then
            $s_i = 0$
            if $b == 0$ then nexticdfa($a - 1, k - 1$)
            else nexticdfa($a, b - 1$)
        else $s_i = s_i + 1$
```

Note that the last string for each sequence of flags has the value $s_l = j$ for $l \in [f_j + 1, f_{j+1} - 1]$, with $j \in [1, n - 1]$. The time complexity of the generator is linear in the number of automata. As an example, for $k = 2$ and $n = 9$ it took about 12 hours to generate all the 705068085303 ICDFA$_\emptyset$'s, using a AMD Athlon at 2.5GHz. Finally, for the generation of ICDFA's we only need to add to the string representation of an ICDFA$_\emptyset$, a string of n 0's and 1's, correspondent to one of the 2^n possible choices of final states.

5 Counting Regular Languages (in Slices)

To obtain the number of languages accepted by DFA's with n states over an alphabet of k symbols, we can generate all ICDFA's, determine which of them are minimal ($f_k(n)$) and calculate the value of $g_k(n)$. Obviously, this is in general an intractable procedure. But for small values of n and k some experiments can take place. We must have an efficient implementation of a minimization algorithm, not because of the size of each automaton but because the number of automata we need to cope with. For that we implemented Hopcroft's minimization algorithm [Hop71], using efficient set representations. For very small values of n and k ($n + k < 16$) we represented sets as bitmaps and for larger values, AVL trees [Avl] were used.

The problem can be parallelized providing that the space search can be safely par-

	n	ICDFA$_\emptyset$	ICDFA	Minimal ($f_k(n)$)	Minimal %	Time (s)
$k = 2$	2	12	48	24	50%	0
	3	216	1728	1028	59%	0.018
	4	5248	83968	56014	66%	0.99
	5	160675	5141600	3705306	72%	79.12
	6	5931540	379618560	286717796	75%	8700
	7	256182290	32791333120	**25493886852**	77%	1237313
$k = 3$	2	56	224	112	50%	0.002
	3	7965	63720	41928	65%	0.7
	4	2128064	34049024	26617614	78%	494.72
	5	914929500	29277744000	**25184560134**	86%	652703
$k = 4$	2	240	960	480	50%	0.01
	3	243000	1944000	1352732	69%	23.5
	4	642959360	10287349760	**7756763336**	75%	184808
$k = 5$	2	992	3968	1984	50%	0.041
	3	6903873	55230984	36818904	66%	756.2

Table 1: Performance and number of minimal automata.

titioned. Using the method presented in Section 4, we can easily generate *slices* of ICDFA$_\emptyset$'s and feed them to the minimization algorithm. A *slice* is a sequence of ICDFA$_\emptyset$'s and is defined by a pair $(start, last)$, where *start* is the first automaton in the sequence and *last* is the last one. If we have a set of CPUs available, each one can receive a slice, generate all ICDFA$_\emptyset$'s (in that slice), generate all the necessary ICDFA's and feed them to the minimization algorithm. For the generation of ICDFA's, we used the observation by Domaratzki *et al.* [DKS02], that is enough to test 2^{n-1} sets of final states, using the fact that a DFA is minimal *iff* its complementary automaton is minimal too. In this way, we can safely divide the search space and distribute each slice to a different CPU. Note that this approach relies in the assumption that we have a much more efficient way to partition the search space than to actually perform the search (in this case a minimization algorithm). The task of creating the slices can be taken by a central process that successively generates the next slice and at the end assembles all the results. The *server* can run interactively with its *slaves*, or it can generate all the slices at once to be used later. The server generates a *slice* using the generator algorithm presented in Section 4. For this experiment we used two approaches. We developed a simple slave management system – called `Hydra` — based on `Python` threads, that was composed by a server and a variable set of *slaves*. In this case, the slaves can be any computer[3]. For each slice a process was executed via `ssh`, and the result was returned to the server. Another approach was to use a computer grid, in particular 24 AMD Opteron 250 2.4GHz (dual core).

5.1 Experimental results

In Table 1, we summarise some experimental results. Most of the values for $k = 2$ and $k = 3$, were already given by Domaratzki *et al.* in [DKS02] and the new results are in bold in the table. For $k = 2, n = 8$ we have divided the universe of ICDFA$_\emptyset$'s in 254 slices and the estimated CPU time for each one to be processed is 11 days.

Moreover, the slicing process can give new insights about the distribution of minimal automata. Figure 2 presents two examples of the values obtained for the rate of minimal DFA's. For $n = 7$ and $k = 2$ we give the percentage of minimal automata for each of the

[3] We used all the normal desktop computers of our colleagues in the CS Department.

257 slices we had used to divide the search space (32791333120 ICDFA$_\emptyset$'s). Each slice had about 100000 ICDFA$_\emptyset$'s, and so 128000000 ICDFA's, and it took about 78 minutes to conclude the process. The whole set of automata was processed in 12 hours of real time of a CPU grid, that corresponds to 344 hours of CPU time.

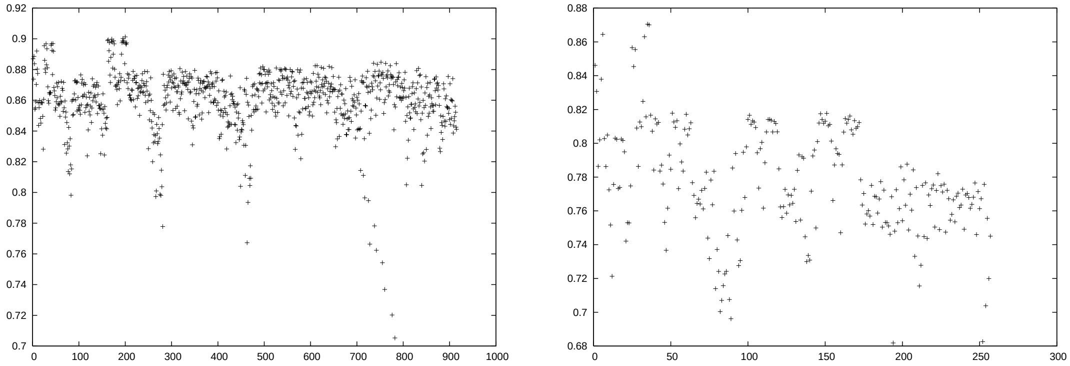


Figure 2: Rate of minimal DFA's with ($k = 3$,$n = 5$) for 915 slices and with ($k = 2$,$n = 7$) for 257 slices.

6 A Uniform Random Generator

The ICDFA$_\emptyset$ representation presented (Section 3) permits an easy random generation for ICDFAs, and thus for DFAs. To randomly generate a DFA for a given n and k, it is only necessary to: (i) randomly generate a valid sequence of flags $(f_i)_{i\in[1,n-1]}$ according to **G1** and **G2**; (ii) followed by the random generation of the rest of the nk elements of the string following **G3**–**G5** rules; (iii) and finally the random generation of the set of final states. The uniformity issue for steps (ii) and (iii) is quite straightforward. For step (iii) it is just necessary to use a uniform random integer generator for a value $i \in [0, 2^n]$. It is enough, for step (ii) the repeated use of the same number generator for values in the range $[0, i]$ for $0 \leq i < n$ according to rules **G3**–**G5**. Step (i) is the only step that needs special care. Consider the case $n = 5$ and $k = 2$. Because of rule **R1** flag f_1 can only be on positions 0 or 1. But there are 140450 ICDFA$_\emptyset$'s with f_1 in the first case and only 20225 in the second. Thus the random generation of flags, to be uniform, must take this into account by making the first case more probable than the second. We can generate a random ICDFA$_\emptyset$ generating its representing string from left to right. Supposing that flag f_{m-1} is already placed at position i and all the symbols to its left are generated, i.e., the prefix $s_0 s_1 \cdots s_i$ is already defined, then the process can be described by:

```
$r = \mathbf{random}(1, \sum_{j=i+1}^{mk-1} N_{m,j})$
for $j = i + 1$ to $mk - 1$:
    if $r \in \left[\sum_{l=i}^{j-1} N_{m,l}, \sum_{l=i}^{j} N_{m,l}\right]$ then return $i$
```

where `random(a,b)` is an uniform random generated integer between `a` and `b`, and $N_{m,j}$ is the number of ICDFA$_\emptyset$s with prefix $s_0 s_1 \cdots s_i$ with the first occurrence of symbol m in position j, making $N_{m,i} = 0$ to simplify the expressions. The values for $N_{m,j}$ could be obtained from expressions similar to Equation (1), and used in a program. But the program would have a exponential time complexity. By expressing $N_{m,j}$ in a recursive form, we have, given k and n

$$\begin{aligned} N_{n-1,j} &= n^{nk-1-j} && \text{with } j \in [n-2, (n-1)k-1]; \\ N_{m,j} &= \sum_{i=0}^{(m+1)k-j-2} (m+1)^i N_{m+1,j+i+1} && \text{with } m \in [1, n-2], \\ & && j \in [m-1, mk-1]. \end{aligned} \tag{2}$$

This evidences the fact that we keep repeating the same computations with very small variations, and thus, if we use some kind of tabulation of this values ($N_{m,j}$), with the obvious price of memory space, we can create a version of a uniform random generator, that apart of a constant overhead used for tabulation of the function refered, has a complexity of $\mathcal{O}(n^3k)\mathcal{O}(\texttt{random})$. The algorithm is described by the following:

```
for $i = (n-1)k-1$ downto $n-2$:
    $N_{n-1,i} = n^{nk-1-i}$
for $m = n-2$ downto $1$:
    $N_{m,mk+1} = \sum_{i=0}^{k-1} (m+1)^i N_{m+1,mk+i}$
    for $i = mk-2$ downto $m-1$:
        $N_{m,i} = (m+1)N_{m,i+1} + N_{m+1,i+1}$
$g = -1$
for $i = 1$ to $n-1$:
    $f$ = generateflag($i, g+1$)
    for $j = g+1$ to $f-1$:
        print random($0, i-1$)
     print $i$
    $g = f$
```

```
def generateflag($m, l$):
    $r$ = random( $0$ , $\sum_{i=l}^{mk-1} m^{i-l} N_{m,i}$ )
    for $i = l$ to $mk-1$:
        if $r < m^{i-l} N_{m,i}$
        then return $i$
        else $r = r - m^{i-l} N_{m,i}$
```

This means that with the same AMD Athlon 64 at 2.5GHz, using a C implementation with `libgmp` [GMP] the times reported in Table 2 were observed. It is possible, without unreasonable amounts of RAM to generate random automata for unusually large values of n and k. For example, with $n = 1000$ and $k = 2$ the memory necessary is less than 450MB. The amount of memory used is so large not only because of the amount of tabulated values, but because the size of the values is enormous. To understand that, it is enough to note that the total number of ICDFA$_\emptyset$'s for these values of n and k is greater than 10^{3350}, and the values tabulated are only bounded by this number.

	$k = 2$	$k = 3$	$k = 5$	$k = 10$	$k = 15$
$n = 10$	0.10s	0.16s	0.29s	0.61s	1.30s
$n = 20$	0.31s	0.49s	1.26s	4.90s	12.24s
$n = 30$	0.54s	1.37s	3.19s	19.91s	62.12
$n = 50$	1.61s	3.86s	17.58s	2.22m	947.71s
$n = 75$	3.96s	12.98s	76.69s	700.20s	2459.34s
$n = 100$	7.92s	36.33s	215.32s	2219.04s	8091.30s

Table 2: Times for the random generation of 10000 automata.

6.1 Statistical test of the random generator

Although the method used to generate random automata is, by its own construction, uniform, we used χ^2 test to evaluate the random generation quality. The universe of ICDFA$_\emptyset$'s with 6 states and 2 symbols has a total size of 5931540. This size is large enough for a test with some significance and it is still reasonable, both in time and space, to perform the test. We generated three different sets of 3000000 ICDFA$_\emptyset$'s and perform the test in each one. Because of the size of the data, we could not find any tabulated values for acceptance, and thus the following formula was used with $v = 30000000 - 1$ and x_p being the significance level (1% in this case):

$$v + 2\sqrt{vx_p} + \frac{3}{4}x_p^2 - \frac{2}{3}.$$

The size of the data sets and the repetition of the test for three times, is the recommended procedure by Knuth ([Knu81], pages 35–39). For the three experiments the values obtained were, respectively, 5933268.92456, 5925676.75108 and 5935733.28172, that are all smaller than the acceptance limit, that for this case was 5938980.75468.

7 Enumeration of ICDFA$_\emptyset$'s

In this section, we show how, given a string representation of an ICDFA$_\emptyset$'s of size n over an alphabet of k symbols, we can compute its number in the generation order (described in Section 4) and vice-versa, i.e., given a number less than $B_{k,n}$, we obtain the corresponding ICDFA$_\emptyset$. This provides an optimal encoding for ICDFA$_\emptyset$'s, as defined by M. Lothaire in [Lot05], Chapter 9. This bijection is accomplished using the tables defined in Section 6 that correspond to partial sums of Equation (1).

Theorem 4 $B_{k,n} = \sum_{l=0}^{k-1} N_{1,l}$.

Proof 2 *The result follows easily by expanding $N_{m,j}$ using Equations (2) and Equation (1).*

7.1 From ICDFA$_\emptyset$'s to Integers

Let $(s_i)_{i\in[0,kn-1]}$ be an ICDFA$_\emptyset$'s string representation, and let $(f_j)_{j\in[1,n-1]}$ be the corresponding sequence of flags. From the sequence of flags we obtain the following number, n_f,

$$n_f \;=\; \sum_{i=1}^{n-1} \sum_{j=f_i+1}^{ik-1} (i^{j-f_i} N_{i,j} (\prod_{m=1}^{i-1} (m^{f_{m+1}-f_m-1})) \tag{3}$$

which is the number of the first ICDFA$_\emptyset$ with flags $(f_j)_{j\in[1,n-1]}$. Now we must add the information provided by the rest of the elements of the string $(s_i)_{i\in[0,kn-1]}$:

$$n_r \;=\; \sum_{j=1}^{n-1} \left(\sum_{l=f_j+1}^{f_{j+1}-1} s_l (j+1)^{f_{j+1}-1-l} \left(\prod_{m=j+1}^{n-1} (m+1)^{f_{m+1}-f_m-1} \right) \right) \tag{4}$$

And the corresponding number is $n_s = n_f + n_r$.

7.2 From Integers to ICDFA$_\emptyset$'s

Given an integer $0 \leq m < B_{k,n}$ a string representing uniquely an ICDFA$_\emptyset$ can be obtained using a method inverse of the one in the last section. The flags $(f_j)_{j\in[1,n-1]}$ are generated from right-to-left, by successive subtractions. The rest of the string $(s_i)_{i\in[0,kn-1]}$ is generate considering the remainders of integer divisions. The algorithms are the following:

```
$s = 1$
for $i$ = 1 to $n-1$:
    $j = i * k - 1$
    $p = i^{j-f_{i-1}-1}$
    while $j >= i-1$ and $m \geq p * s * N_{i,j}$:
        $m = m - N_{i,j} * p * s$
        $j = j - 1$
        $p = p/i$
    $s = s * i^{j-f_{i-1}-1}$
    $f_i = j$
```

```
$i = k * n - 1$
$j = n - 1$
while $m > 0$ and $j > 0$:
    while $m > 0$ and $i > f_j$:
        $s_i = m \mod (j+1)$
        $m = m \div (j+1)$
        $i = i - 1$
    $i = i - 1$
    $j = j - 1$
```

8 Final Remarks

The methods here presented were implemented and tested to obtain both exact and approximate values for the density of minimal automata. Champarnaud *et al.* in [CP05], checked a conjecture of Nicaud that for $k = 2$ the number of minimal ICDFA's is about 80% of the total, by sampling automata with 100 states (for all possible number of final states). Our results also corroborate that conjecture, being the exact values for some small values of n and samples for greater values. In particular, for $k = 2$ and $n = 100$ we obtained the same results as Champarnaud *et al.*. It seems that for $k > 2$ almost all ICDFA's are minimal. For $k = 3, 5$ and $n = 100$ that was also checked by Champarnaud *et al.*. For a confidence interval of 99% and significance level of 1% the following table presents the percentages of minimal ICDFA's for several values of k and n, and each possible number of final states.

$k\backslash n$	5	6	7	8	9	10	20	40	80	160
3	85.8%	90.8%	93.3%	95.0%	96.1%	96.7%	98.7%	99.4%	99.7%	99.8%
5	93.0%	96.5%	98.2%	99.1%	99.5%	99.8%	100.0%	100.0%	100.0%	100.0%
7	93.7%	96.8%	98.4%	99.2%	99.6%	99.8%	100.0%	100.0%	100.0%	–
9	93.7%	96.9%	98.4%	99.2%	99.6%	99.8%	100.0%	100.0%	–	–
11	93.8%	96.9%	98.4%	99.2%	99.6%	99.8%	100.0%	100.0%	–	–
13	93.7%	96.9%	98.4%	99.2%	99.6%	99.8%	100.0%	100.0%	–	–

A web interface to the random generator can be found in the **FAdo** project web page [Fad]. Bassino and Nicaud in [BN] presented also a random generator of ICDFA's based on Boltzmann Samplers, recently introduced by Duchon *et al.* [DFLS04]. However the sampler is uniform for partitions of a set with kn elements into n nonempty subsets (not for the universe of automata). These partitions are related with string representations that verify only rule **R1**. Based on the work here presented, it would be interesting to study a better approximation, that would satisfy rule **R2**.

9 Acknowledgments

We thank the anonymous referees for their comments that helped to improve this paper.

References

[Avl] Gnu Libavl, binary search trees library. http://www.stanford.edu/∼blp/avl/.

[BN] Frédérique Bassino and Cyril Nicaud. Enumeration and random generation of accessible automata. Submitted. `http://www-igm.univ-mlv.fr/∼bassino/publi.html`

[CP05] Jean-Marc Champarnaud and Thomas Paranthoën. Random generation of DFAs. *Theoretical Computer Science*, 330(2):221–235, 2005.

[DFLS04] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability & Computing*, 13(4-5):577–625, 2004.

[DKS02] Michael Domaratzki, Derek Kisman, and Jeffrey Shallit. On the number of distinct languages accepted by finite automata with n states. *Journal of Automata, Languages and Combinatorics*, 7(4):469–486, 2002.

[Fad] FAdo: tools for formal languages manipulation. http://www.ncc.up.pt/fado.

[GMP] GNU multi precision arithmetic library. http://www.swox.com/gmp/.

[Hop71] John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Proc. Inter. Symp. on the Theory of Machines and Computations*, pages 189–196, Haifa, Israel, 1971. Academic Press.

[Knu81] Donald E. Knuth. *The Art of Computer Programming. Seminumerical Algorithms.*, volume 2. Addison Wesley, 2nd edition, 1981.

[Lot05] M. Lothaire. *Applied Combinatorics on Words.* Cambridge Univ, Press, 2005.

[Nic00] Cyril Nicaud. *Étude du comportement en moyenne des automates finis et des langages rationnels.* PhD thesis, Université de Paris 7, 2000.

[Par04] Thomas Paranthoën. *Génération aléatoire et structure des automates à états finis.* PhD thesis, Université de Rouen, 2004.

[RMA05] Rogério Reis, Nelma Moreira, and Marco Almeida. On the representation of finite automata. In C. Mereghetti, B. Palano, G. Pighizzini, and D.Wotschke, editors, *7th International Workshop on Descriptional Complexity of Formal Systems*,pages 269–276, Como, Italy, June 2005.

[SF96] Robert Sedgewick and Philippe Flajolet. *Analysis of Algorithms.* Addison-Wesley, 1996.

[Slo03] Neil Sloane. The On-line Encyclopedia of Integer Sequences, 2003. http://www.research.att.com/∼njas/sequences.